

**RECEIVED  
CENTRAL FAX CENTER****JUL 17 2006**

Appl. No. 10/682,387  
Response to 1<sup>st</sup> Office Action dated 07/17/2006  
Reply to 1<sup>st</sup> Office Action of 04/17/2006

**REMARKS**

In this Office Action, the Examiner rejected Claims 1 – 20 under 35 U.S.C. §102(b) as being anticipated by Bak et al.

Examiner Syed and primary Examiner Mofiz are thanked for the telephone interview of July 13, 2006. In that interview, Applicants' attorney and the two Examiners discussed Claim 1 and the applied reference. Applicants' attorney pointed out to Examiner Mofiz the passage on page 7, lines 23 - 30 of the Specification as support for the claim. Examiner Mofiz suggested that if the passage is included in the claim, the claim will not be anticipated by the reference. After reviewing the claim, it seems that the only thing from that passage missing in the claim is the word "QUEUE." However, since the terms "putting a thread in the queue of CPU<sub>1</sub>" and "scheduling the thread to run on CPU<sub>1</sub>" are equivalent, applicants' attorney did not think that the claim needed to be amended to include the word QUEUE.

However, Applicants' attorney amended the independent claims to better claim the invention. Thus, by this amendment, Claims 1 – 20 remain pending in the Application. For the reasons stated more fully below, Applicant's attorney submits that the claims are allowable over the applied references. Hence, reconsideration, allowance and passage to issue are respectfully requested.

As mentioned in the SPECIFICATION, when a process is being processed by a CPU and for some reason needs to wait for an event to occur before proceeding, for efficiency reasons, the process may cede the rest of its turn at the CPU to another process and goes to sleep. If the process has a lock on a shared resource, it will not relinquish the lock before it goes to sleep. For example, when a first process is using a shared resource such as a buffer, it will put a lock on the buffer to prevent all other processes from using the buffer. If the first process was performing some disk input/output (I/O), it may allow another process to use the CPU and go to sleep while the disk I/O is completing. Once the disk I/O has completed, the first process may awaken. If a second

AUS920030439US1

Appl. No. 10/682,387

Response to 1<sup>st</sup> Office Action dated 07/17/2006

Reply to 1<sup>st</sup> Office Action of 04/17/2006

process with a higher priority number needs to use the buffer in the mean time, it will have to wait until the first process obtains some CPU time to complete its task and release the lock on the buffer.

To reduce the amount of time the second process may have to wait, priority boosting has been used. Priority boosting occurs when a second process with a higher priority number passes its priority number to a first process with a lower priority number and which has a lock on a needed shared resource to increase the first process' likelihood at being the next process chosen to receive some CPU time. As will be explained later, although it now has a higher priority number, the first process may not obtain the CPU right away if another process is currently using the CPU. Thus, what is needed is a method of enhancing priority boosting such that a process that has a lock on a shared resource and whose priority has been boosted may obtain some CPU time as soon as possible.

The present invention provides such a method. According to the teachings of the invention, when a second thread being executed on a second CPU determines that it has to wait for a lock on a shared resource held by a first thread that is scheduled to be executed by a first CPU, if the second thread has a higher priority than the first thread, it will boost the priority of the first thread by passing its higher priority to the first thread. After doing so, the second thread will enhance the priority boost of the first thread by rescheduling the first thread to run on the second CPU.

By rescheduling the first thread to run on the second CPU, the second thread is ensuring that the first thread will have CPU time right away since in essence the second thread is giving its turn at the CPU to the first thread.

The invention is set forth in claims of varying scopes of which Claim 1 is illustrative.

1. A method of enhancing priority boosting of scheduled threads comprising the steps of:  
determining by a second thread being executed on  
a second CPU whether to wait for a lock on a shared

AUS920030439US1

Appl. No. 10/682,387

Response to 1<sup>st</sup> Office Action dated 07/17/2006

Reply to 1<sup>st</sup> Office Action of 04/17/2006

resource held by a first thread that is scheduled to be executed by a first CPU, the second thread having a higher priority than the first thread;

boosting the priority of the first thread by passing the higher priority of the second thread to the first thread if the second thread has to wait for the lock on the shared thread; and

***enhancing the priority boosting of the first thread by rescheduling the first thread to run on the second CPU.*** (Emphasis added.)

The Examiner rejected all the claims in the Application under 35 U.S.C. §102(b) as being anticipated by Bak et al. Applicant respectfully disagrees.

Bak et al. purport to teach a method of locking and unlocking objects by threads. According to the teachings of Bak et al., a first thread is used to obtain a header value of an object by replacing the contents of a header field of the object with a sentinel that identifies an execution stack associated with the first thread. Once the object header field contents are replaced with the sentinel, a determination is made regarding whether the object header field contents include a header value of the object. If it is determined that the object header field contents do not include the header value of the object, a determination is made as to when the object is in the process of being studied by a second thread. If it is determined that the object is not in the process of being studied by a second thread, the method involves adding the first thread to a list associated with the stack associated with the second thread. The list is arranged to indicate that the first thread is awaiting access to the object.

If, on the other hand, it is determined that the object is in the process of being studied by a second thread, the object header field contents are resolved to identify the second thread. And, it is determined whether an execution priority associated with the second thread is less than an execution priority associated with the first thread. If it is determined that the second thread execution priority is less than the first thread execution priority, the method further includes boosting the second thread execution priority to match the first thread execution priority.

AUS920030439US1

Appl. No. 10/682,387

Response to 1<sup>st</sup> Office Action dated 07/17/2006

Reply to 1<sup>st</sup> Office Action of 04/17/2006

However, Bak et al. does not teach, show or so much as suggest the step of ***enhancing the priority boosting of a first thread by rescheduling the first thread to run on the second CPU*** as claimed.

The Examiner asserted that the passages in col. 23, lines 15 – 35 and in col. 24, lines 55 – 57 “clearly illustrates an example of rescheduling a thread to run on the CPU.”

However, it should be noted that the step of enhancing the priority boost is not merely rescheduling a boosted thread but rather rescheduling the boosted thread from CPU<sub>1</sub> to CPU<sub>2</sub> for example (i.e., taking the thread from the run queue of CPU<sub>1</sub> and putting it in the run queue of CPU<sub>2</sub>), see page 7, lines 23 – 30, page 8, lines 20 – 26, Figs. 1(a) – 1(f) and elements 212 and 214 of Fig. 2 of the present Application.

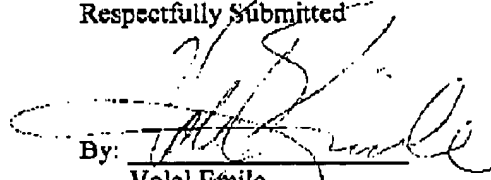
Nowhere in their disclosure do Bak et al. disclose the step of enhancing a thread boost by rescheduling the boosted thread such that it can be processed by a CPU other the one on which it has originally been scheduled to run.

Consequently, Applicant submits that Claim 1, as well as its dependent claims should be allowable. The other independent claims (i.e., Claims 6 and 11 and 16), which all incorporate the emboldened-italicized limitations in the above-reproduced Claim 1 and their dependent claims should be allowable as well. Hence, Applicant once more requests reconsideration, allowance and passage to issue of the claims in the application.

AUS920030439US1

Appl. No. 10/682,387  
Response to 1<sup>st</sup> Office Action dated 07/17/2006  
Reply to 1<sup>st</sup> Office Action of 04/17/2006

Respectfully Submitted



By:

Volel Emile  
Attorney for Applicants  
Registration No. 39,969  
(512) 306-7969

AUS920030439US1

Page 11 of 11